TP Grenouille, pliage et monade

1. Introduction : Kermit monte les escaliers

Kermit, un mâle grenouille, cherche à rejoindre la femelle Erane au sommet d'un escalier. Pour ce faire, il saute de marche en marche, mais le nombre de marche dont il réussit à monter est quelque peu aléatoire...

Pour représenter les déplacements de Kermit, on définit une fonction, `saut(marche: int) -> [int]` qui, pour un niveau donné (on commence au niveau 0, avant la première marche)), donne la liste des marches accessibles après un saut.

Proposer différentes versions de cette fonction dans les cas suivants :

1. `saut1` : Kermit avance aléatoirement d'une ou 2 marche
2. `saut2` : Même chose, mais l'escalier a une hauteur fixée (20 marches), et Kermit ne peut donc dépasser le niveau 20
3. `saut3` : Sur les niveaux pairs, Kermit peut avancer d'une ou 2 marches, sur les niveaux impairs, Kermit peut avancer de 0, 1 ou 2 marches.

2. Enchaîner les sauts...

En partant du niveau 0, on aimerait savoir où Kermit peut se trouver après :

- un saut : il suffit d'appliquer la fonction `saut` ;
- 2 sauts : on peut utiliser la fonction `map` ;
- 3 sauts: on ne peut plus utiliser `map` car après 2 sauts, on n'a plus une liste, mais une liste de listes.

On va donc définir la fonction `flat_map` : cette fonction doit fonctionner un peu comme la fonction `map`, sauf qu'elle prend en premier paramètre une fonction qui renvoie une liste et, elle doit *aplatir* le résultat, en ne renvoyant pas une liste de listes, mais une simple liste.

Exemple : on suppose que la fonction `step`, correspond au cas où Kermit avance de 1 ou 2 marches.

```
>>> map(step, [0, 1])`
[[1,2], [2, 3]]
>>> flat_map(step, [0, 1])
[1, 2, 2, 3]
```

Remarque : une même position peut apparaître plusieurs fois ; cela permet d'avoir une idée de la probabilité qu'à Kermit de se trouver sur une case donnée.

1. Définir la fonction `flat_map` comme on le sent.
2. Si ce n'est pas ce qui a été fait, définir la fonction `flat_map` grâce à la fonction `reduce` du module *functools*. S'assurer qu'un seul parcours de

liste est effectué.

3. Écrire la fonction `repeter_sauts(positions: list[int], nb: int) -> list[int]` qui génère la liste des positions où peut se trouver la grenouille après *nb* sauts (de type `saut3`) à partir d'une liste de positions potentielles.
4. Écrire la fonction d'ordre supérieur `repeter_sauts_parametre(int -> list[int], list[int], nb: int) -> list[int]` dont le premier paramètre est la fonction décrivant le saut unitaire de Kermit.

3. Éviter les doublons

Dans certains cas, il peut être souhaitable de ne pas garder plusieurs occurrences d'un même niveau.

1. Écrire une fonction `flat_map_uniq(t -> list[t], iter[t])` qui fait la même chose que `flat_map`, mais sans doublon.
2. Écrire la fonction d'ordre supérieur `repeter_sauts_parametre_uniq(int -> list[int], list[int], nb: int) -> list[int]` qui fait la même chose que `repeter_saut_parametre`, mais sans doublon.

4. Pour aller plus loin avec les pliages...

4.1 Calcul de fréquence

On souhaite maintenant calculer la liste des couples (niveau, nb_occurrences) pour savoir, après n déplacements, le nombre de cas où Kermit peut être à un niveau donné.

1. Écrire une classe `Frequence` encapsulant un dictionnaire (valeur, frequence) avec :

- une méthode `ajouter(valeur)` qui permet d'ajouter une occurrence de *valeur* et qui renvoie l'instance courante de `Frequence` ;
- une méthode `to_liste()` qui renvoie la liste des couples correspondant aux différentes entrées du dictionnaire.

2. Écrire la fonction `to_freq(liste[int]) -> liste[(int, int)]` en utilisant la fonction `reduce` et la classe `Frequence` pour obtenir, à partir d'une liste de position avec multi-occurrences, la liste des couples (position, fréquence). L'accumulateur utilisé par le `reduce` sera ici une instance de `Frequence`, qui joue le rôle de **collecteur** : il collecte les informations nécessaires au fur et à mesure que le `reduce` parcours la liste (grâce à la méthode `ajouter`).

4.2 Calculs statistiques

On souhaite calculer des statistiques diverses sur une liste de couples (position, valeur). Ici, on se limitera au calcul de la position moyenne.

1. Écrire une classe `Statistiques` proposant les méthodes suivantes :

- `ajouter(valeur, poids = 1)` ; comptabilise (au fur et à mesure) le fait qu'on a ajouté `poids` données pour une valeur totale de `valeur * poids` Cette méthode doit également renvoyer l'instance courante de la classe `Statistiques` ;
- `moyenne() -> float` : permet de connaître la moyenne des données analysées ;
- `nombre()-> int`: permet de connaître le nombre de données analysées.

2. Écrire la fonction `pos_moyenne(list[(int, int)]) -> float` qui permet de calculer la position moyenne de Kermit. Cette fonction fera appel à `reduce` et utilisera une instance de la classe `Statistiques` en guise de *collecteur*.